Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем
Системное программирование

Шигаров Никита Алексеевич

Среда визуального программирования роботов на .NET

Выпускная квалификационная работа

Научный руководитель:
к.т.н., доц. Литвинов Ю. В.

Рецензент:
д.ф.-м.н., проф. Терехов А. Н.

Санкт-Петербург
2017

SAINT-PETERSBURG STATE UNIVERSITY

Software and Administration of Information Systems
Software Engeneering

Shigarov Nikita

Visual programming environment for robots on .NET

Graduation Project

Scientific supervisor:
C.Sc. Docent Yurii Litvinov

Reviewer:
Ph.D. Professor Andrew Terekhov

Saint-Petersburg
2017

Оглавление

1. Введение

Предметно-ориентированные языки (Domain Specific Language или DSL) очень важны в современном мире программирования. Они служат удобным средством разработки архитектуры, бизнес-процессов и т. д. Также существуют визуальные предметно-ориентированные языки, которые в некоторых важных случаях оказываются удобнее текстовых. Например, они полезны для обучения детей составлению алгоритмов, так как в данном случае изучение синтаксиса языка может быть затруднительно.

На кафедре системного программирования СПбГУ создан проект QReal [14] — кроссплатформенный свободно распространяемый под лицензией Apache License 2.0[1] инструмент с открытым исходным кодом, предназначенный для создания специализированных сред визуального программирования.

Этот инструмент используется в ядре среды визуального программирования роботов TRIK Studio, которая позволяет составлять визуальные схемы алгоритмов, генерировать по ним код в несколько языков программирования, симулировать поведение роботов, а также отправлять программу на роботы Lego NXT, Lego EV3 или ТРИК [1]. TRIK Studio разработана на языке C++ с использованием библиотеки Qt [8].

Недавно стартовал новый проект REAL.NET [12], являющийся идейным продолжателем QReal и реализуемый на платформе .NET. В связи с этим возник интерес изучения технологии Modeling SDK, которая позволяет создавать собственные визуальные языки как плагины к Visual Studio [6], как аналога разрабатываемого REAL.NET с целью выяснить его сильные и слабые стороны, а также написать код, потенциально переиспользуемый в будущей реализации среды программирования роботов в REAL.NET.

[1]Домашняя страница проекта QReal, URL: https://github.com/qreal/qreal (дата обращения: 21.05.2017)

2. Постановка задачи

Целью данной работы является разработка прототипа среды визуального программирования роботов ТРИК на .NET с помощью Modeling SDK, генерирующей код на JavaScript и использующей визуальный язык, близкий к языку среды TRIK Studio. Для достижениия этой цели были сформулированы следующие задачи:

- выполнить обзор возможностей среды программирования роботов TRIK Studio;
- исследовать структуру Modeling SDK;
- создать архитектуру решения;
- реализовать решение;
- выполнить апробацию.

3. Существующие используемые решения

3.1. Обзор системы TRIK Studio

TRIK Studio представляет из себя визуальную среду программирования роботов Lego Mindstorms NXT, Lego Mindstorms EV3 и ТРИК, которая позволяет генерировать код на языках C#, F#, JavaScript, Pascal.ABC в случае конструктора ТРИК, NXT OSEK C в случае Lego NXT и байткод виртуальной машины в случае Lego EV3 по графическим диаграммам [15].

Визуальный язык среды TRIK Studio [13] устроен следующим образом. Различные управляющие блоки соединены направленными связями переходов, задающих порядок выполнения инструкций. У блоков и связей могут быть заданы свои параметры. Так, у блока, отвечающего за конструкцию «Условие», может быть задан параметр условия, а у блока, отвечающего за мощность моторов, параметры мощности и номеров портов, на которые нужно подать мощность. Из двух исходящих связей, отвечающих за ветки блока «Условие», одна должна быть помечена символом «истина». Каждая из исходящих связей, отвечающих за ветки блока «Выбор», должна быть помечена соответствующей меткой. Также должны быть помечены идентификаторами потоков каждая исходящая связь блока, отвечающего за создание параллельных задач. Циклы задаются как замкнутые цепочки.

Программа на визуальном языке может быть либо отлажена на компьютере (с опциональной посылкой пакетов на реального робота), либо сгенерирована в текстовый код и загружена на робота. В первом случае пользователь может создать виртуальный двухмерный мир, состоящий из стенок, цветных элементов и размеченных регионов и сэмулировать поведение робота в данной среде. Во втором случае визуальная программа генерируется в текстовый файл и отображается в текстовом редакторе, основанном на библиотеке qscintilla[2], а затем отправляется на робота.

[2]Домашняя страница библиотеки qscintilla, URL: https://www.riverbankcomputing.com/software/qscintilla/intro (дата обращения: 22.05.2017)

Пользовательский интерфейс среды TRIK Studio представлен на рис. 1.

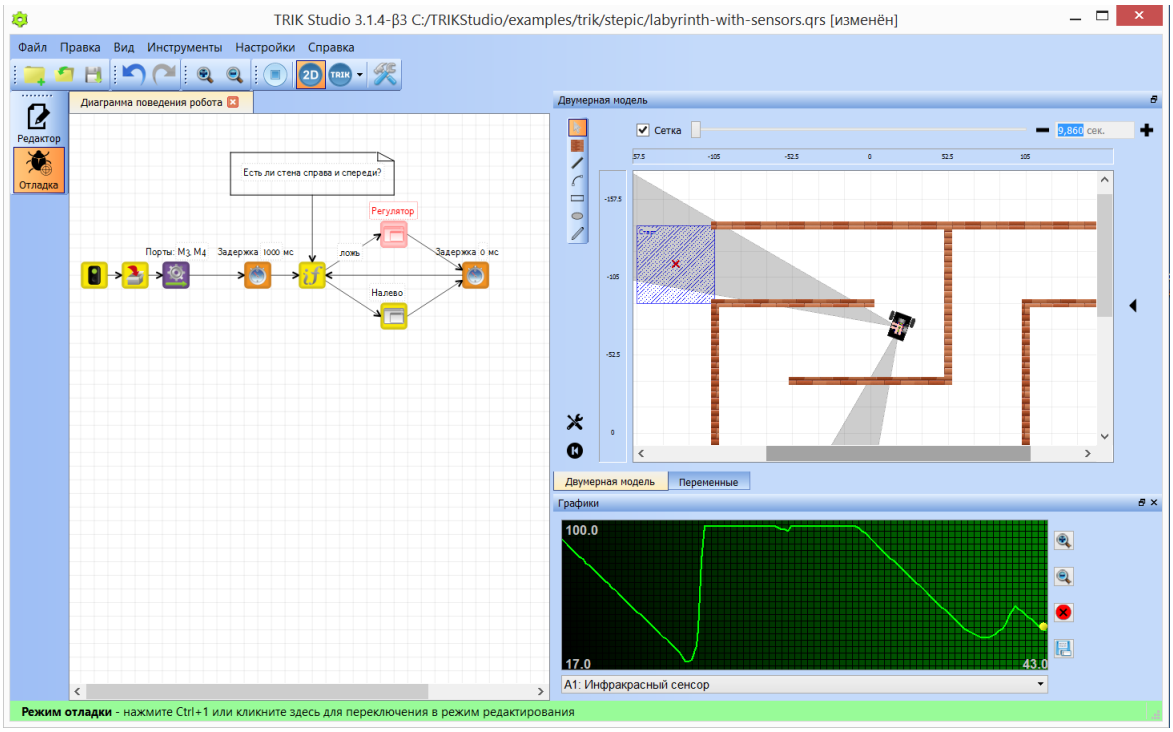

Рисунок 1. Редактор TRIK Studio

При разботке среды ипользовался язык C++ и библиотека Qt. Таким образом, она является кроссплатформенной. Среда бесплатна, имеет открытый исходный код под лицензией Apache License 2.0.

3.2. Обзор технологии Modeling SDK

Modeling SDK for Visual Studio является платформой по разработке визуальных предметно-ориентированных языков, которая позволяет интегрировать в Visual Studio свои визуальные предметно-ориентированные языки программирования. До Visual Studio 2017 предоставлялась как подключаемый модуль, начиная с 2017 версии интегрирована в среду [6].

Разработка визуального предметно-ориентированного происходит следующим образом (см. рис. 2). Создается проект по разработке DSL, затем происходит создание и редактирование метамодели (описание визуального языка, то есть множества всех синтаксически корректных

диаграмм на этом языке), генерация реализационных классов, компиляция и отладка DSL-пакета в новом экземпляре Visual Studio.

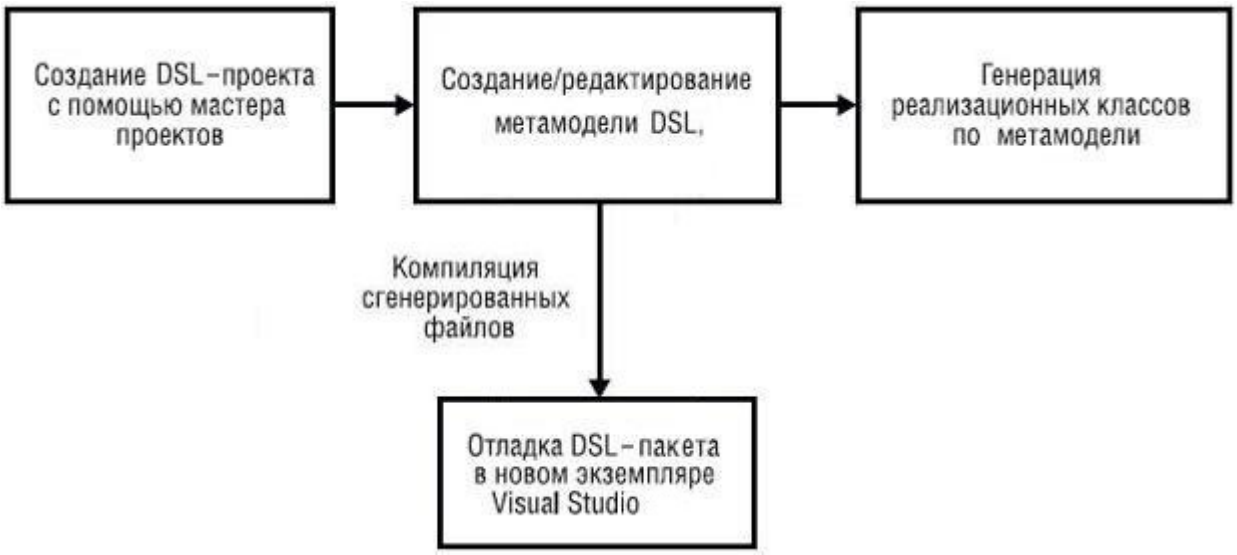

Рисунок 2. Процесс разработки DSL

Для программирования метамодели используется графический редактор Modeling SDK, но также можно переопределить или добавить новые методы в сгенерированные частичные классы языка C# вручную.

Для генерации используется язык описания шаблонов T4 [3], который представляет из себя совокупность управляющих команд на языках C# или Visual Basic, выделямых символами “<#” и “#>”, и текстовых блоков.

Шаблон преобразуется в программу на соответствующем языке, результатом работы которой будет код целевой системы.

В новом решении автоматически создаются проект Dsl и DslPackage [11]. Первый предназначен для хранения различных артефактов метамодели создаваемого DSL, второй хранит настройки пользовательского интерфейса целевого графического редактора. В обоих проектах можно добавлять расширения классов на языке C# для расширения базовой функциональности нового редактора.

Для хранения моделей используются доменные классы — абстракции сущностей хранимых данных в модели. Так же есть классы фигур,

свойства которых связаны со свойствами классов моделей посредством механизма событий в языке C# [9]. Эти классы представляют собой абстрации элементов диаграммы и их свойства видны пользователю в текстовых полях блоков на диаграмме. Можно задавать отношения между классами через создание специальных классов, тем самым обеспечивая задание правил для построения визуальных схем.

Способы представления и хранения моделей уже реализованы в SDK стандартными средствами языка C#.

Механизмы валидации моделей также реализованы в SDK. Любой доменный класс можно пометить атрибутом, который наделяет класс возможностью быть валидированным. После этого, нужно создать метод класса со своим специальным атрибутом, который указывает, при каких случаях будет запускаться этот метод — по вызову из меню, при сохранении или открытии модели, или из пользовательского кода. После выявления ошибок корректности свойств соответствующей модели, если такие имеются, нужно записать их в логгер ошибок соответствующего объекта ValidationContext, который хранит информацию о текущем процессе валидации.

4. Архитектура решения

Для решения поставленных задач была разработана следующая архитектура (см. рис. 3).

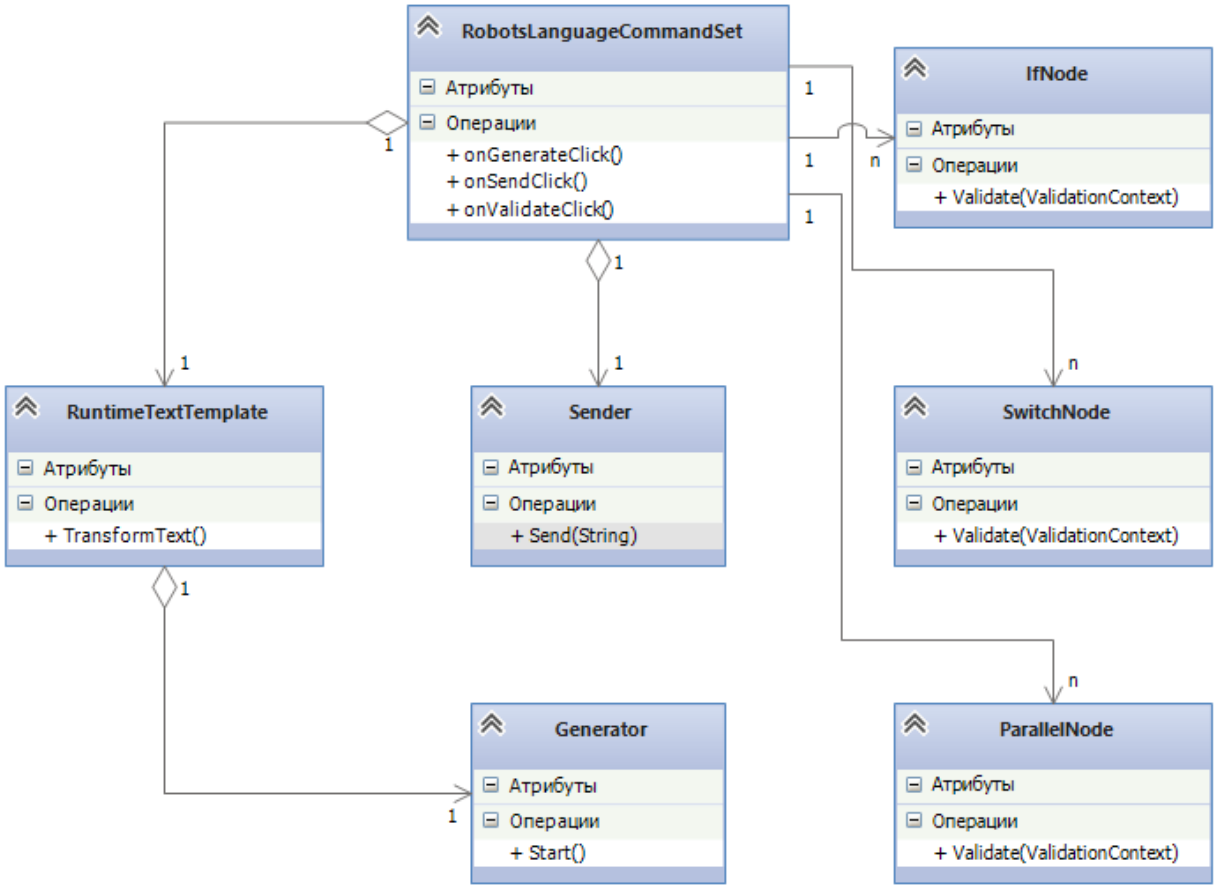

Рисунок 3. Диаграмма классов системы

Редактор должен позволять строить как корректные, так и некорректные диаграммы на языке, приближенном к визуальному языку среды TRIK Studio. Никаких ограничений на структуру диаграммы не налагается в целях удобства пользователя.

Проверку корректности должен выполнять валидатор, запускаемый автоматически перед генерацией кода, либо по желанию пользователя.

Генератор кода в классе Generator должен генерировать код, который впоследствие либо будет записан в файл с последующим открытием в редакторе Visual Studio, либо отправлен на робота по каналу

TCP/IP. Метод генерации запускается из класса шаблона RuntimeTextTemplate с передачей текущего экземпляра этого класса для вызова метода сохранения сгенерированной информации в этом объекте.

Валидатор проверяет корректность расстановки связей для блоков разного типа, меток на связях и блоках, а также корректность структуры диаграммы. Этот код должен быть реализован в методе соответствующего доменного класса с атрибутом валидации.

Варианты выполнить валидацию, генерацию в файл, либо отправку на робота выбираются пользователем в меню. Обработчики данных пунктов рализованы в классе RobotsLanguageCommandSet.

5. Особенности реализации

5.1. Редактор кода

Для того, чтобы реализовать корректное составление диаграмм, была создана метамодель, приведенная на рис. 5. Существует базовый абстрактный класс Compound, представляющий собой составной объект, который может содержать в себе другие узлы. От него наследуется главная модель RobotModel, представляющая основную диаграмму поведения робота, и SubprogramNode — класс, котрый представляет собой подпрограмму, которая может быть вызвана из основной программы или подпрограмм. Compound может содержать несколько AbstractNode (класс, являющийся базовым для всех классов элементов модели) и SubprogramNode. AbstractNode в свою очередь может ссылаться на AbstractNode и от AbstractNode наследуются все конкретные доменные классы, что позволяет всем элементам диаграммы иметь возможность ссылаться друг на друга. Таким образом, можно в модели строить схемы, состоящие из этих элементов. При этом добавление нового элемента будет производиться довольно просто. У разных доменных классов есть свои свойства: так, например, у классов IfNode и SwitchNode, соответствующих блокам «Если» и «Выбор», есть свойства Condition, которые хранят строки условия. Для каждого конкретного класса есть соответствующий класс фигуры типа ImageShape, значения свойств которых связаны друг с другом. Соответственно, количество текстовых полей, выводящих информацию о свойствах, соответствующего доменного класса, совпадает с количеством этих свойств.

Также были программно переопределены свойства AllowsChildrenToResizeParent и MinimumResizableSize в классе фигуры Subprogram, соответствующей элементу «Субпрограмма» для того, чтобы динамически изменять размер элемента при добавлении новых элементов на него [7].

Были реализованы следующие конструкции:

- алгоритмические: Start (начало выполнения), Finish (конец вы-

полнения), If, Switch, While, For, Break, Parallel Tasks (параллельные задачи), Subprogram Call (вызов подпрограмм), Subprograms (субпрограммы, представленные как отдельные контейнеры);

- системные: Motors (подача мощности на моторы), MotorsStop (остановка моторов), Delay (задержка), WaitTouch (ожидание датчика касания), WaitSensor (ожидание датчика расстояния).

Получившийся визуальный редактор представлен на рис. 4.

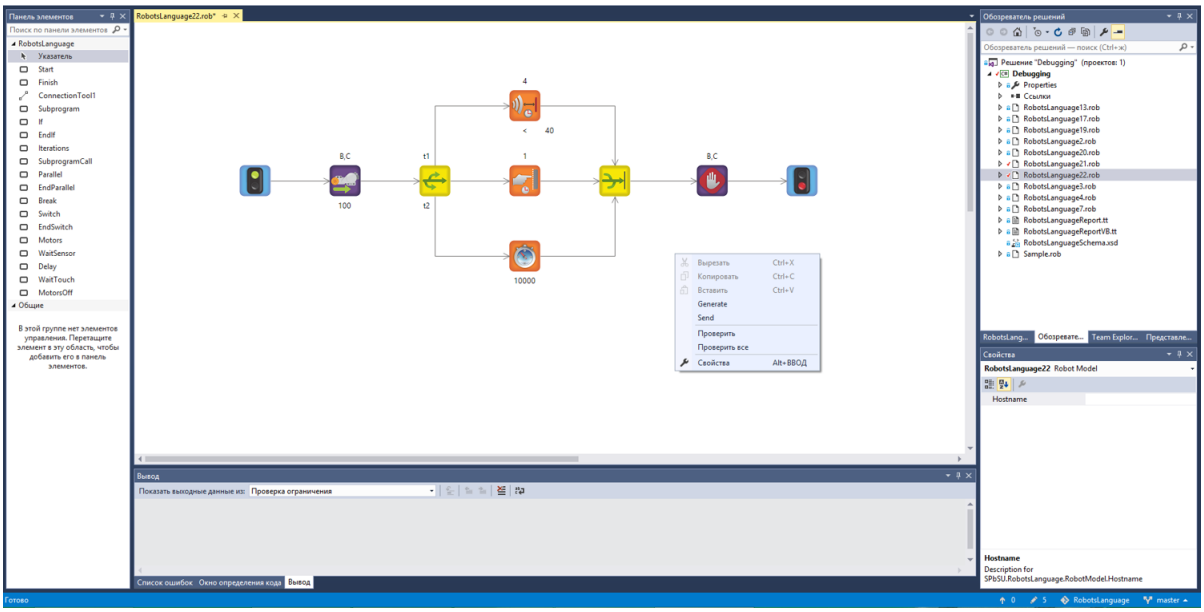

Рисунок 4. Окно визуального редактора

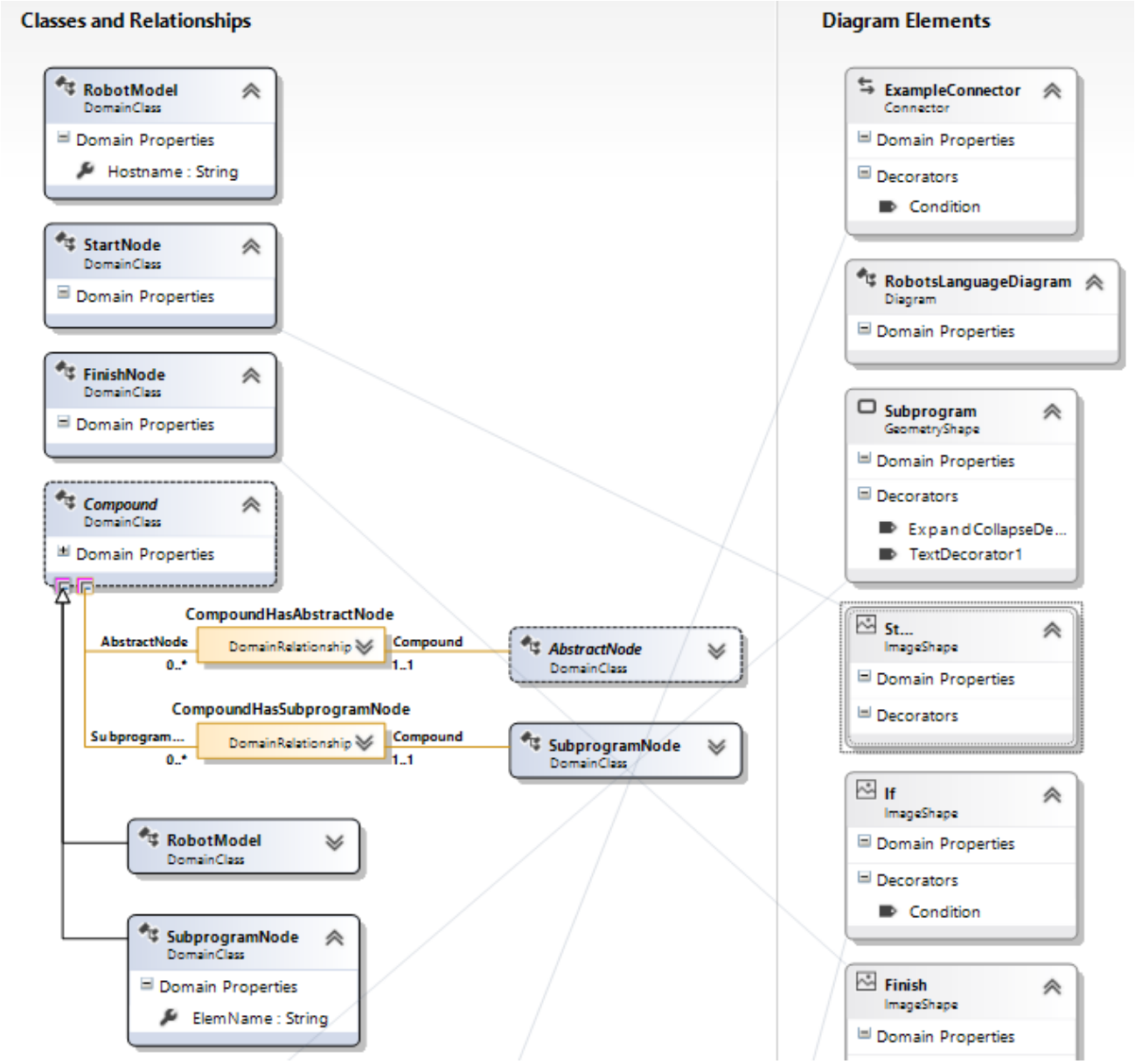

Рисунок 5. Метамодель языка моделирования

5.2. Валидация кода

Доменный класс главной модели был расширен валидирующим методом, который обходит коллекцию AbstractNode и проверяет корректность количества входящих и исходящих связей, а также меток на них. Затем запускается модифицированный алгоритм генератора в данном решении (о котором будет рассказано далее), для того, чтобы проверить корректность структуры диаграммы. В этой модификации мы не генерируем код и дополнительно проверяем, что ветки разветвляющих

блоков сходятся в один и тот же конечный блок.

Ошибки выводятся в логгер ошибок, соответствующего объекта ValidationContext, информация из которого выводится в окно ошибок Visual Studio. При клике на ошибку происходит подсветка соответствующего элемента. Окно ошибок представлено на рис. 6.

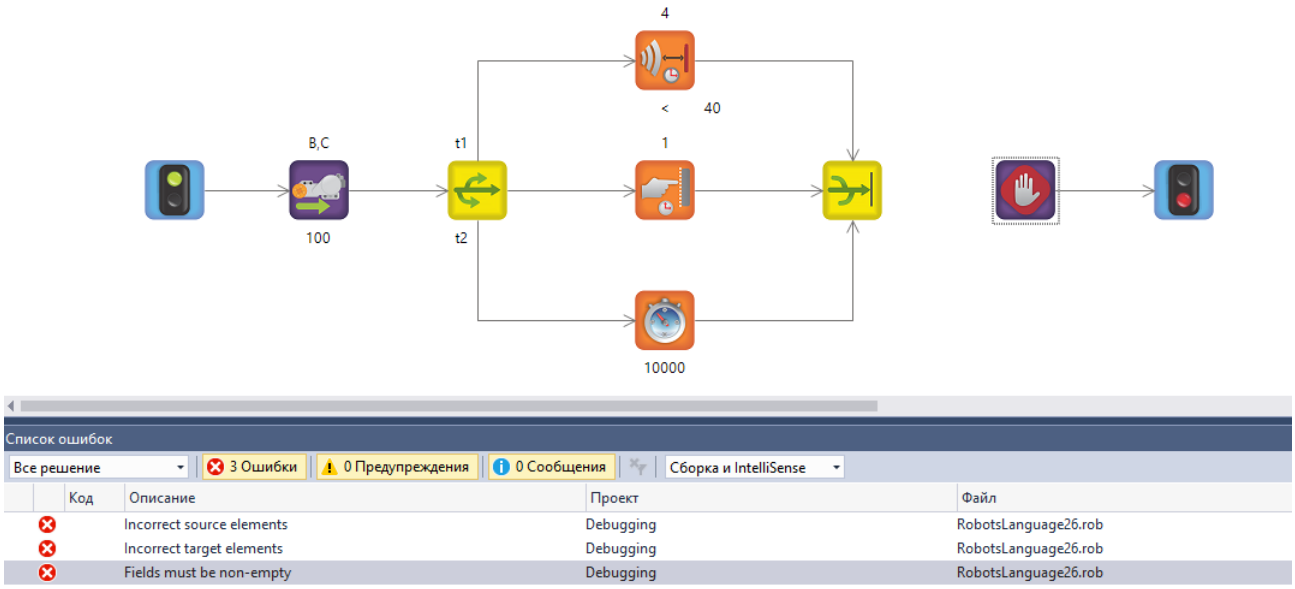

Рисунок 6. Диаграмма и окно ошибок

5.3. Генерация кода

Для ралиазации генератора был создан текстовый шаблон RuntimeTextTemplate, в котором в управляющем блоке создается объект генератор с передачей объектов модели и текущего контекста (для сохранения генерируемых строчек в классе генерации в объекте класса этого шаблона) и вызывается метод генерации. Модель передается в класс шаблона при обработке нажатия пункта меню.

Алгоритм генерации заключается в следующем (см. Algorithm 1). Ищем стартовый блок и запускаем рекурсивную функцию генерации с этого блока, передавая имя соответствующего конечного блока — финиш. Аналогично на блоки If, Switch, параллельные задачи мы передаем соотвествующие им конечные блоки. Если в текущий блок есть дополнительная входящая связь, то значит, что мы обнаружили цикл.

Как в случае с циклом, так и в случае блока итераций мы передаем в функцию идентификатор начального блока, как конечный. В функции генерации мы в цикле переходим по блокам, пока не встретим соответствующий конечный блок. Таким образом, мы легко узнаем структуру программного кода: в начале функции пишется начало алгоритмической конструкции, потом происходит рекурсивный вызов функции генерации от следующего блока и по её завершении мы дописываем конец соответствующего алгоритмической конструкции.

Субпрограммы заранее генерируются в отдельные функции тем же алгоритмом, а при встрече блоков вызова субпрограмм мы просто вызываем соответствующее имя функции. В случае блока параллельных задач мы сначала генерируем ветки кода, не совпадающие с текущим потоком, в отдельные функции, запускаем их при генерации и переходим к следующему блоку текущего потока.

В случае остальных блоков генерируются фиксированные строчки кода с подстановкой соответствующих параметров, задаваемых на блоках.

На рис. 7 представлена диаграмма, на которой происходит подача мощности на моторы, затем запуск потоков, в одном из которых происходит ожидание датчика расстояния с указанной дистанцией, в другом — остановка потока на заданное время, а в основном — ожидание датчика касания. Затем основной поток ожидает завершения двух других и останавливает моторы.

На рис. 8 представлен сгенерированный файл по диаграмме, представленной на рис. 7.

Algorithm 1 Функция генерации

```
1: function GENERATE(currentBlock, endBlock)
2:     while currentBlock <> endBlock do
3:         if ISCYCLEBLOCK(currentBlock) then
4:             condition ← GETCONDITIONPARAMETER(IfBlock)
5:             trueBlock ← GETTRUENEXTBLOCK(currentBlock)
6:             falseBlock ← GETFALSENEXTBLOCK(currentBlock)
7:             WRITE(if (condition))
8:             GENERATE(trueBlock, EndIfBlock)
9:             WRITE(else)
10:            currentBlock ← GENERATE(falseBlock, EndIfBlock)
11:        else if currentBlock = IfBlock then
12:            condition ← GETCONDITIONPARAMETER(currentBlock)
13:            startBlock ← GETNEXTCYCLEBLOCK(currentBlock)
14:            WRITE(while (condition))
15:            currentBlock ← GENERATE(startBlock, currentBlock)
16:        else if currentBlock = motorsBlock then
17:            ports ← GETPORTSPARAMETER(currentBlock)
18:            power ← GETPOWERPARAMETER(currentBlock)
19:            for all port in ports do
20:                WRITE(brick.motor(port).setPower(power))
21:            end for
22:            currentBlock ← GETNEXTBLOCK(currentBlock)
23:        else if currentBlock = ParallelBlock then
24:            threads ← GETTHREADLINKS(currentBlock)
25:            for all thread in threads do
26:                WRITE(Threading.startThread(thread))
27:            end for
28:            currentBlock ← GETCURRENTTHREADBLOCK(currentBlock)
29:        else if currentBlock = EndParallelBlock then
30:            newThread ← GETTHREADNAME(GETNEXTLINK(currentBlock))
31:            if thread = newThread then
32:                threads ← GETTHREADLINKS(currentBlock)
33:                for all thread in threads do
34:                    WRITE(Threading.joinThread(thread))
35:                end for
36:                currentBlock ← GETNEXTBLOCK(currentBlock)
37:            else
38:                WRITE(return)
39:            end if
40:        end if
41:    end while
42:    return GETNEXTBLOCK(currentBlock)
43: end function
```

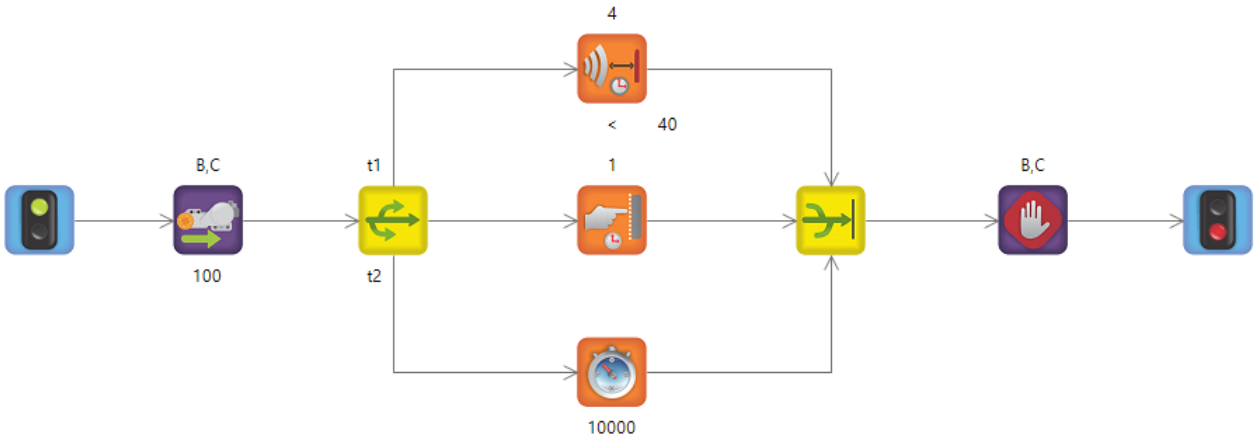

Рисунок 7. Диаграмма параллельных задач

RobotsLanguage21.js RobotsLanguage21.rob*

<глобальные>

```
1
2  function ParallelNode1_0() {
3      while (!(brick.sensor(4).read() < 40)) {
4          script.wait(10);
5      }
6      return;
7  }
8  function ParallelNode1_1() {
9      script.wait(10000);
10     return;
11 }
12 function main() {
13     brick.motor(B).setPower(100);
14     brick.motor(C).setPower(100);
15     Threading.startThread("t1", "ParallelNode1_0");
16     Threading.startThread("t2", "ParallelNode1_1");
17     while (brick.sensor(1).read() < 0) {
18         script.wait(10);
19     }
20     brick.motor(B).powerOff();
21     brick.motor(C).powerOff();
22     return;
23 }
24
```

Рисунок 8. Сгенерированный файл

Для автоматического открытия сгенерированного файла в Visual Studio была использована оболочка COM-библиотеки EnvDTE [2].

Отправка программы при выборе соответствующего пункта меню происходит по TCP-соединению [5] на IP-адрес, указанный в свойстве модели.

6. Апробация решения

Произведена апробация решения путем составления различных диаграмм и тестировании на реальном роботе.

Был проведен юзабилити-эксперимент на нескольких студентах, которым предлагалось составить программу для робота, в которой при обнаружении стены, он должен отъезжать и подъезжать к ней снова и снова. После решения этой задачи предлагалось заполнить анкету System Usability Scale [10] и оставить отзыв.

Анкета System Usability Scale представляет надежный инструмент для измерения удобства пользования приложением. Она состоит из 10 вопросов с пятью вариантами ответов для респондентов от полного согласия до полного несогласия. Эти вопросы приведены ниже.

1. Я бы хотел(-а) поработать еще с этой программой
2. Программа слишком сложная
3. Этой программой легко пользоваться
4. Мне понадобится помощь, чтобы научиться пользоваться этой программой
5. Разные функции в этом приложении правильно сгруппированы
6. В приложении слишком много несоответствий
7. Большая часть людей очень быстро научиться пользоваться этой программой
8. Это приложение очень трудно использовать
9. Я уверенно себя чувствовал(-а), используя это приложение
10. Мне пришлось многому научиться, прежде чем я смог(-ла) работать с приложением

В зависимости от степени согласия с вопросами 1, 3, 5, 7, 9 начисляется от 0 (полное несогласие) до 4 (полное согласие) баллов. За вопросы 2, 4, 6, 8, 10 от 4 (полное несогласие) до 0 баллов (полное согласие). Затем сумма очков домножается на 2.5 для получения номированного относительно 100 результата. Исследования показали, что средний балл находится в районе 68 баллов.

Результаты тестирования составили 70, 65, 73, 60 и 80 баллов.

Одна из составленных диаграмм пользователей представлена на рис. 9.

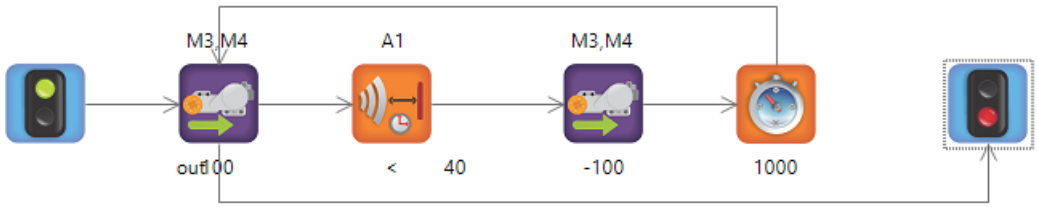

Рисунок 9. Составленная диаграмма пользователем

7. Обсуждение

Визуальный редактор данного решения может быть знаком пользователям по среде Visual Studio. Это является плюсом для профессиональных разработчиков, студентов или старшеклассников, имеющих опыт работы в этой среде разработки, но может быть плохо для школьников младших классов, поскольку интерфейс Visual Studio ориентирован на разработчиков и может быть сложен для детей.

Благодаря Modeling SDK многие вещи, такие как хранение моделей или визуальная отрисовка связей, уже реализованы на хорошем уровне. Так, например, благодаря автоматической отрисовке связей, обходящих другие элементы, связи между блоками при пересечении выглядят корректно.

Отдельный валидатор является плюсом данной среды, в отличие от TRIK Studio, где запускается сразу генерация кода с открытием нового файла.

Поскольку для создания прототипа системы не было необходимости перереализовывать алгоритм генерации, используемый в TRIK Studio, алгоритм генерации, реализованный в данном решении, проще, чем в TRIK Studio. Из-за этого построение диаграмм в данном решении отличается от TRIK Studio тем, что нужно обязательно ставить конечные блоки для разветвляющих и проводить выходящую связь из цикла с меткой out. Однако любую диаграмму TRIK Studio можно в том или ином виде представить в данном решении.

Относительно долгий процесс установки и запуска Visual Studio может быть расценен как минус данной системы. Однако с правильной настройкой запуска изолированной среды (отключение ненужных компонентов) скорость запуска может быть увеличена [4].

8. Заключение

В ходе данной работы получены следующие результаты:

- выполнен обзор возможностей среды программирования роботов TRIK Studio;
- выполнен обзор Modeling SDK for Visual Studio;
- разработана архитектура решения;
- реализована визуальная среда программирования роботов;
- произведена апробация решения путем тестирования на реальном роботе.

Список литературы

[1] A. Kuzenkova A. Deripaska T. Bryksin Y. Litvinov V. Polyakov. QReal DSM Platform: An Environment for Creation of Specific Visual IDEs. — Proceedings of 8th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2013), SCITEPRESS, 2013, pp. 251-257.

[2] Cheda K. Greenwood J. Bischof B. Pro Visual Studio .NET. — Apress; Softcover reprint of the original 1st ed. edition (August 30, 2004).

[3] Code Generation and T4 Text Templates. — https://msdn.microsoft.com/en-us/en-en/library/bb126445.aspx (дата обращения: 21.05.2017г).

[4] Customizing the Isolated Shell. — https://msdn.microsoft.com/en-us/en-en/library/ee390883.aspx (дата обращения: 21.05.2017г).

[5] David B. Makofske Michael J. Donahoo Kenneth L. Calvert. TCP/IP Sockets in C#: Practical Guide for Programmers. — Morgan Kaufmann.

[6] Modeling SDK for Visual Studio - Domain-Specific Languages. — https://msdn.microsoft.com/en-us/library/bb126259.aspx (дата обращения: 21.05.2017г).

[7] Prieur Jean-Marc. [DSL Tools] Support of Nested shapes in Visual Studio 2008 SP1. — https://blogs.msdn.microsoft.com/jmprieur/2008/09/03/dsl-tools-support-of-nested-shapes-in-visual-studio-2008-sp1/ (дата обращения: 21.05.2017г).

[8] Qt Documentation. — http://doc.qt.io/ (дата обращения: 21.05.2017г).

[9] Richter Jeffrey. CLR via C#: (4th Edition). — Microsoft Press; 4 edition, P. 249.

[10] Sauro Jeff. A Practical Guide to the System Usability Scale: Background, Benchmarks and Best Practices. — CreateSpace Independent Publishing Platform.

[11] Steve Cook Gareth Jones Stuart Kent Alan Wills. Domain-specific development with visual studio dsl tools. — Addison-Wesley Professional © 2007.

[12] Литвинов Ю.В. Кузьмина Е.В. Небогатиков И.Ю. Алымова Д.А. Среда предметно-ориентированного визуального моделирования REAL.NET. — https://github.com/yurii-litvinov/articles/blob/master/2017-realNet/realNet.pdf (дата обращения: 21.05.2017г).

[13] Мордвинов Д.А. Литвинов Ю.В. Новые возможности среды программирования роботов TRIK Studio. — VI Всероссийская конференция «Современное технологическое обучение: от компьютера к роботу» (сборник тезисов), СПб., ЗАО «Полиграфическое предприятие № 3», 2016. С. 41-43.

[14] Терехов А.Н. Брыксин Т.А. Литвинов Ю.В. QReal: платформа визуального предметно-ориентированного моделирования. — Программная инженерия, 2013, № 6, С. 11-19.

[15] Ю.В. Литвинов. Реализация визуальных средств программирования роботов для изучения информатики в школах. — Компьютерные инструменты в образовании, СПб., 2013, № 1, С. 36-45.